项目工程搭建

一. 创建Flink 工程

```
mvn archetype:generate                               \
      -DarchetypeGroupId=org.apache.flink              \
      -DarchetypeArtifactId=flink-quickstart-java      \
      -DarchetypeVersion=1.10.0

groupId: com.nx
artifactId: flink-ad
version: 1.0-SNAPSHOT
package: com.nx.stream
```

```
vim pom.xml
<flink.version>1.7.2</flink.version>
```

二. 定义Schema

数据交换格式 选型 Xml、Json、Protobuf

1. 在main下面创建 proto 文件夹
2. ad_server_log.proto

```
syntax = "proto3";//请在非空非注释的第一行指定当前文件使用的是proto3的语法,默认proto2
package grpc;//package与java_package有些不同, java_package是定义编译生成的java文件所在的目录, 而package是对应的java类的命名空间
option java_package = "com.nx.stream.entity";
option java_outer_classname = "AdServerLogProto";//要生成Java类的名称
option java_multiple_files = true;//编译后会生成多个Message类

message AdServerLog{
    string request_id = 1;
    uint64 timestamp = 2;

    // client info
    string device_id = 11;
    string os = 12;
    string network = 13;

    // dmp info
    string user_id = 51;
    string gender = 52;
    int32 age = 53;
    string country = 54;
    string province = 55;
    string city = 56;

    // ad info
    string source_type = 100;
    string bid_type = 101;
    int64 bid_price = 102;

    int64 pos_id = 150;
    int64 account_id = 151;
    int64 unit_id = 152;
    int64 creative_id = 153;
    // event
    string event_type = 200;

}
```

3. ad_client_log.proto

```
syntax = "proto3";//请在非空非注释的第一行指定当前文件使用的是proto3的语法,默认proto2
package grpc;//package与java_package有些不同，java_package是定义编译生成的java文件所在的目录，而package是对应的java类的命名空间
option java_package = "com.nx.stream.entity";
option java_outer_classname = "AdClientLogProto";//要生成Java类的名称
option java_multiple_files = true;//编译后会生成多个Message类

message AdClientLog{
    string request_id = 1;
    uint64 timestamp = 2;

    string device_id = 11;
    string os = 12;
    string network = 13;

    // dmp info
    string user_id = 51;

    // ad info
    string source_type = 100;
    string bid_type = 101;

    int64 pos_id = 150;
    int64 account_id = 151;
    int64 unit_id = 152;
    int64 creative_id = 153;
    // event
    string event_type = 211;
}
```

4. ad_log.proto

```
syntax = "proto3";//请在非空非注释的第一行指定当前文件使用的是proto3的语法,默认
proto2
package grpc;//package与java_package有些不同，java_package是定义编译生成的java
文件所在的目录，而package是对应的java类的命名空间
option java_package = "com.nx.stream.entity";
option java_outer_classname = "AdLogProto";//要生成Java类的名称
option java_multiple_files = true;//编译后会生成多个Message类

message ProcessInfo {
    bool join_server_log = 1;
    int64 process_timestamp = 2;
    int64 retry_count = 3;
}

message AdLog{
    string request_id = 1;
    uint64 timestamp = 2;
    ProcessInfo process_info = 3;

    string device_id = 11;
    string os = 12;
    string network = 13;

    // dmp info
    string user_id = 51;
    string gender = 52;
    int32 age = 53;
    string country = 54;
    string province = 55;
    string city = 56;

    // ad info
    string source_type = 100;
    string bid_type = 101;
    int64 bid_price = 102;

    int64 pos_id = 150;
    int64 account_id = 151;
    int64 creative_id = 152;
    int64 unit_id = 153;
    // event
    string event_type = 200;
    int64 send = 201;
    int64 impression = 202;
    int64 click = 203;
    int64 download = 204;
    int64 installed = 205;
    int64 pay = 206;
}
```

三. 添加proto构建插件

vim pom.xml # 插入到 build/plugins 下面

```
            <!--添加编译proto文件的编译程序和对应的编译插件-->
            <plugin>
                <groupId>org.xolstice.maven.plugins</groupId>
                <artifactId>protobuf-maven-plugin</artifactId>
                <version>0.5.0</version>
                <configuration>

<protoSourceRoot>${project.basedir}/src/main/proto</protoSourceRoot>

<outputDirectory>${project.basedir}/src/main/java</outputDirectory>
                    <clearOutputDirectory>false</clearOutputDirectory>

<protocArtifact>com.google.protobuf:protoc:3.1.0:exe:${os.detected.classifier}
</protocArtifact>
                    <pluginId>grpc-java</pluginId>
                    <pluginArtifact>io.grpc:protoc-gen-grpc-
java:1.14.0:exe:${os.detected.classifier}</pluginArtifact>
                </configuration>
                <executions>
                    <execution>
                        <goals>
                            <goal>compile</goal>
                            <goal>compile-custom</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
```

vim pom.xml # 插入到 build 下面

```
        <extensions>
            <extension>
                <groupId>kr.motd.maven</groupId>
                <artifactId>os-maven-plugin</artifactId><!--引入操作系统os设置的属
性插件,否则${os.detected.classifier} 操作系统版本会找不到 -->
                <version>1.5.0.Final</version>
            </extension>
        </extensions>
```

vim pom.xml # 插入到 dependency 下面

```
        <dependency>
            <groupId>com.google.protobuf</groupId>
            <artifactId>protobuf-java</artifactId>
            <version>3.4.0</version>
            <!--            <version>2.5.0</version>-->
        </dependency>
```

Mock 脚本开发

一. 创建kafka

```
cd /work/hadoop/kafka_2.12-2.4.0
./bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor
1 --partitions 2 --topic server_log_v2
./bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor
1 --partitions 2 --topic client_log_v2
./bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor
1 --partitions 2 --topic client_log_retry_v2
./bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor
1 --partitions 2 --topic ad_log_v2
./bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor
1 --partitions 2 --topic ad_log_reprot_v2
```

确保kafka创建成功：http://10.0.0.9:9666/clusters/NXKafka/topics

二. 开发MockDataUtils

在main下面创建`package com.nx.stream.mock;`

Vim MockDataUtils.java

```
   static String[] provinceArray = {"北京", "上海", "重庆", "天津", "香港", "澳
门", "台湾"};

    public static String gender() {
        double random = Math.random();
        if (random > 0.4)
            return "男";
        else
            return "女";
    }

    public static int age() {
        double random = Math.random() * 70;
        return (int) (random % 100) + 18;
    }

    public static String country() {
        double random = Math.random();
        if (random > 0.13)
            return "中国";
        else
            return "其他";
    }

    public static String province() {
        double random = Math.random() * 17;
        return provinceArray[(int) (random % provinceArray.length)];
    }

    public static long bidPrice() {
        double random = Math.random() * 100000;
        return (long) (random % 100000);
    }

//    String requestId, String deviceId, String os, String network, String
userId, String sourceType,
//    String bidType, long posId, long accountId, long unitId, long
creativeId,
    public static String requestId() {
        return RandomStringUtils.random(15, "abcdefgABCDEFG123456789");
    }

    public static String deviceId() {
        return RandomStringUtils.random(10, "abcdefgABCDEFG123456789");
    }

    public static String os() {
        double random = Math.random();
        if (random > 0.4)
            return "IOS";
        else
            return "ANDROID";
    }
    public static String network() {
        double random = Math.random();
        if (random > 0.4)
            return "4G";
        else if (random > 0.1)
            return "WIFI";
        else
            return "3G";
    }

    public static String userId() {
        return RandomStringUtils.random(8, "abcdefgABCDEFG123456789");
    }

    public static String sourceType() {
        double random = Math.random();
        if (random > 0.3)
            return "DSP";
        else
            return "ADX";
    }

    public static String bidType() {
        double random = Math.random();
        if (random > 0.4)
            return "CPC";
        else if (random > 0.13)
            return "CPM";
        else
            return "CPA";
    }

    public static long posId() {
        double random = Math.random();
        if (random > 0.74)
            return 5;
        else if (random > 0.33)
            return 9;
        else
            return 17;
    }

    public static long accountId() {
        double random = Math.random();
        if (random > 0.54)
            return 88888;
        else if (random > 0.33)
            return 66666;
        else
```

```
            return 33333;
    }

    public static long unitId(long accountId) {
        int prefix = new Random(accountId).nextInt(100000);
        return prefix * 10 + new Random().nextInt(5);
    }

    public static long creativeId(long unitId) {
        int prefix = new Random(unitId).nextInt(100000);
        return prefix * 10 + new Random().nextInt(10);
    }
```

三. 开发MockData

在main下面创建`package com.nx.stream.mock;`

Vim MockData.java

Mock server log

```
    public void mockServerLog(String requestId, String deviceId, String os,
String network, String userId, String sourceType,
                          String bidType, long posId, long accountId, long
unitId, long creativeId, String eventType) {
        AdServerLog serverLog = AdServerLog.newBuilder()
                .setRequestId(requestId)
                .setTimestamp(System.currentTimeMillis())
                .setDeviceId(deviceId)
                .setOs(os)
                .setNetwork(network)
                .setUserId(userId)
                .setSourceType(sourceType)
                .setBidType(bidType)
                .setPosId(posId)
                .setAccountId(accountId)
                .setUnitId(unitId)
                .setCreativeId(creativeId)
                .setEventType(eventType)

                .setGender(MockDataUtils.gender())
                .setAge(MockDataUtils.age())
                .setCountry(MockDataUtils.country())
                .setProvince(MockDataUtils.province())
                .setBidPrice(MockDataUtils.bidPrice())

                .build();
        ETLUtils.sendKafka(producer, Constants.SERVER_LOG,
serverLog.toByteArray());
```

```
    }
```

ETLUtils

```


    public static void sendRetry(Producer producer, byte[] value) {
        sendKafka(producer, RETRY_TOPIC, value);
    }

    public static void sendKafka(Producer producer, String topic, byte[]
value) {
        ProducerRecord<String, byte[]> record = new ProducerRecord<String,
byte[]>(topic, value);
        producer.send(record);
    }
```

```
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerRecord;
```

```
    <dependency>
      <groupId>org.apache.flink</groupId>
      <artifactId>flink-connector-kafka-0.10_${scala.binary.version}
</artifactId>
      <version>${flink.version}</version>
    </dependency>
```

Constants

```
vim com.nx.stream.utils.Constants
public class Constants {
    public static String SERVER_LOG = "server_log_v1";
    public static String CLIENT_LOG = "client_log_v1";
    public static String CLIENT_LOG_RETRY = "client_log_retry_v1";
    public static String AD_LOG = "ad_log_v1";
    public static String AD_LOG_REPORT = "ad_log_report_v1";
    public static String TABLE_NAME = "context_v1";
    public static String HDFS_LOG_HOME = "/home/";
    public static String BROKERS = "10.0.0.9:9092,10.0.0.8:9092";
}
```

KafkaProducerUtils

```
package com.nx.stream.utils;

import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
```

```java
import java.util.Properties;

public class KafkaProducerUtils {

    static Producer<String, String> producer;

    public static void init(){
        Properties props = new Properties();
        //此处配置的是kafka的端口
        props.put("metadata.broker.list", Constants.BROKERS);
        props.put("bootstrap.servers", Constants.BROKERS);

        //配置value的序列化类
        props.put("value.serializer",
"org.apache.kafka.common.serialization.ByteArraySerializer");
        //配置key的序列化类
        props.put("key.serializer",
"org.apache.kafka.common.serialization.ByteArraySerializer");
        props.put("producer.type","async");


        //request.required.acks
        //0, which means that the producer never waits for an acknowledgement
from the broker (the same behavior as 0.7). This option provides the lowest
latency but the weakest durability guarantees (some data will be lost when a
server fails).
        //1, which means that the producer gets an acknowledgement after the
leader replica has received the data. This option provides better durability
as the client waits until the server acknowledges the request as successful
(only messages that were written to the now-dead leader but not yet replicated
will be lost).
        //-1, which means that the producer gets an acknowledgement after all
in-sync replicas have received the data. This option provides the best
durability, we guarantee that no messages will be lost as long as at least one
in sync replica remains.
        props.put("request.required.acks","-1");

        producer = new KafkaProducer<>(props);

    }


    public static Producer getProducer() {
        if (producer == null) {
            init();
        }
        return producer;
    }
}
```

Mock client log

```
   public void mockClientLog(String requestId, String deviceId, String os,
String network, String userId, String sourceType,
                          String bidType, long posId, long accountId, long
unitId, long creativeId, String eventType ) {
        AdClientLog clientLog = AdClientLog.newBuilder()
                .setRequestId(requestId)
                .setTimestamp(System.currentTimeMillis())
                .setDeviceId(deviceId)
                .setOs(os)
                .setNetwork(network)
                .setUserId(userId)
                .setSourceType(sourceType)
                .setBidType(bidType)
                .setPosId(posId)
                .setAccountId(accountId)
                .setUnitId(unitId)
                .setCreativeId(creativeId)
                .setEventType(eventType)
                .build();
        ETLUtils.sendKafka(producer, Constants.CLIENT_LOG,
clientLog.toByteArray());
    }
```

Mock 逻辑

```
    int magicNum = 30;

    public void mock() {
        String requestId = MockDataUtils.requestId();
        String deviceId = MockDataUtils.deviceId();
        String os = MockDataUtils.os();
        String network = MockDataUtils.network();
        String userId = MockDataUtils.userId();
        String sourceType = MockDataUtils.sourceType();
        String bidType = MockDataUtils.bidType();
        long posId = MockDataUtils.posId();
        long accountId = MockDataUtils.accountId();
        long unitId = MockDataUtils.unitId(accountId);
        long creativeId = MockDataUtils.creativeId(unitId);
        mockServerLog(requestId, deviceId, os, network, userId, sourceType,
bidType, posId, accountId, unitId, creativeId, "SEND");
        if (isImpression(creativeId)) {
            mockClientLog(requestId, deviceId, os, network, userId,
sourceType, bidType, posId, accountId, unitId, creativeId, "IMPRESSION");
            if (isClick(creativeId)) {
                mockClientLog(requestId, deviceId, os, network, userId,
sourceType, bidType, posId, accountId, unitId, creativeId, "CLICK");
                if (isDownload(creativeId)) {
                    mockClientLog(requestId, deviceId, os, network, userId,
sourceType, bidType, posId, accountId, unitId, creativeId, "DOWNLOAD");
                    if (isinstalled(creativeId)) {
                        mockClientLog(requestId, deviceId, os, network,
userId, sourceType, bidType, posId, accountId, unitId, creativeId,
"INSTALLED");
                        if (isPayed(creativeId)) {
                            System.out.println("mock pay");
                            mockClientLog(requestId, deviceId, os, network,
userId, sourceType, bidType, posId, accountId, unitId, creativeId, "PAY");
                        }
                    }
                }
            }
        }

    }

    public boolean isImpression(long creativeId) {
        int base = new Random(creativeId).nextInt(10);
        return new Random().nextInt(37 + base) < base + magicNum;
    }

    public boolean isClick(long creativeId) {
        int base = new Random(creativeId).nextInt(10);
        return new Random().nextInt(47 + base) < base + magicNum;
    }

    public boolean isDownload(long creativeId) {
        int base = new Random(creativeId).nextInt(10);
        return new Random().nextInt(57 + base) < base + magicNum;
    }

    public boolean isinstalled(long creativeId) {
        int base = new Random(creativeId).nextInt(10);
        return new Random().nextInt(67 + base) < base + magicNum;
    }


    public boolean isPayed(long creativeId) {
        int base = new Random(creativeId).nextInt(10);
        return new Random().nextInt(83 + base) < base + magicNum;
    }


    public static void main(String[] args) throws InterruptedException {
        MockData mockData = new MockData();
        while(true) {
            mockData.mock();
            Thread.sleep(100);
        }
    }
```

四. 打包测试

```
mvn clean package
scp -P20522 /Users/liuhongliang/zz/flink-ad/target/flink-ad-1.0-SNAPSHOT.jar
root@116.213.207.93:~
scp flink-ad-1.0-SNAPSHOT.jar 10.0.0.9:/work/hadoop/v1
java -cp flink-ad-1.0-SNAPSHOT.jar:/work/hadoop/druid-0.16.0/lib/*
com.nx.stream.mock.MockData
cd /work/hadoop/kafka_2.12-1.1.0
bin/kafka-console-consumer.sh --bootstrap-server 10.0.0.9:9092,10.0.0.8:9092
--topic server_log_v1 --from-beginning
bin/kafka-console-consumer.sh --bootstrap-server 10.0.0.9:9092,10.0.0.8:9092
--topic client_log_v1 --from-beginning
```

Flink 任务开发

一. Stream join

1. StreamJoinJob - 创建kafka source

```
vim com.nx.stream.ad.StreamJoinJob
// 常量
  private static String KAFKA_SERVER_LOG = Constants.SERVER_LOG;
  private static String KAFKA_CLIENT_LOG = Constants.CLIENT_LOG;
  private static String KAFKA_AD_LOG = Constants.AD_LOG;
  private static String KAFKA_AD_LOG_REPORT = Constants.AD_LOG_REPORT;
  private static String BROKERS = Constants.BROKERS;

  public static void main(String[] args) throws Exception {

    String groupId = "flink-join-test2";
    // set up the streaming execution environment
    final StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
```

```
//    final StreamExecutionEnvironment env =
StreamExecutionEnvironment.createLocalEnvironment();
    env.enableCheckpointing(60000);
    env.setParallelism(2);
    env.getCheckpointConfig().setFailOnCheckpointingErrors(false);
    env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);
    env.getConfig().enableForceAvro();
    env.getConfig().registerTypeWithKryoSerializer(AdServerLog.class,
ProtobufSerializer.class);
    env.getConfig().registerTypeWithKryoSerializer(AdLog.class,
ProtobufSerializer.class);

    Properties properties =
FlinkKafkaConsumerUtils.getConsumerProperties(BROKERS, KAFKA_SERVER_LOG,
groupId);
    Properties properties2 =
FlinkKafkaConsumerUtils.getConsumerProperties(BROKERS, KAFKA_CLIENT_LOG,
groupId);
    // 添加数据源
    DataStreamSource<AdServerLog> adServerInputStream = env.addSource(
        new FlinkKafkaConsumer010<>(KAFKA_SERVER_LOG, new AdServerLogSchema(),
properties));
    DataStreamSource<AdClientLog> adClientInputStream = env.addSource(
        new FlinkKafkaConsumer010<>(KAFKA_CLIENT_LOG, new AdClientLogSchema(),
properties2));
```

FlinkKafkaConsumerUtils

```
    public static Properties getConsumerProperties(String brokers, String
topic, String groupId) {
        Properties properties = getCommonProperties(topic + groupId);
        properties.setProperty("bootstrap.servers", brokers);
        return properties;
    }

    public static Properties getCommonProperties(String groupId) {
        Properties properties = new Properties();
        properties.setProperty("client.id", String.format("consumer-%s-%d",
groupId, System.currentTimeMillis()));
        properties.setProperty("group.id", groupId);
        properties.setProperty("max.partition.fetch.bytes", "3145728");
        properties.setProperty("auto.offset.reset", "latest");
        properties.setProperty("heartbeat.interval.ms", "10000");
        properties.setProperty("flink.partition-discovery.interval-millis",
"60000");
        properties.setProperty("session.timeout.ms", "60000");
        properties.setProperty("request.timeout.ms", "65000");
        properties.setProperty("enable.auto.commit", "false");
        properties.setProperty("auto.commit.interval.ms", "30000");
        return properties;
    }
```

ProtobufSerializer

```
import com.twitter.chill.protobuf.ProtobufSerializer;
        <dependency>
            <groupId>com.twitter</groupId>
            <artifactId>chill-protobuf</artifactId>
            <version>0.9.2</version>
        </dependency>
```

AdServerLogSchema

```
public class AdServerLogSchema implements DeserializationSchema<AdServerLog>,
SerializationSchema<AdServerLog> {
    @Override
    public AdServerLog deserialize(byte[] bytes) throws IOException {
        return AdServerLog.parseFrom(bytes);
    }

    @Override
    public boolean isEndOfStream(AdServerLog adServerLog) {
        return false;
    }

    @Override
    public byte[] serialize(AdServerLog adServerLog) {
        return new byte[0];
    }

    @Override
    public TypeInformation<AdServerLog> getProducedType() {
        return TypeInformation.of(new TypeHint<AdServerLog>() {
        });
    }
}
```

AdClientLogSchema

```
public class AdClientLogSchema implements DeserializationSchema<AdClientLog>,
SerializationSchema<AdClientLog> {
    @Override
    public AdClientLog deserialize(byte[] bytes) throws IOException {
        return AdClientLog.parseFrom(bytes);
    }

    @Override
    public boolean isEndOfStream(AdClientLog adClientLog) {
        return false;
    }

    @Override
    public byte[] serialize(AdClientLog adClientLog) {
        return new byte[0];
    }

    @Override
    public TypeInformation<AdClientLog> getProducedType() {
        return TypeInformation.of(new TypeHint<AdClientLog>() {
        });
    }
}
```

2. StreamJoinJob - 处理ad_server_log

```
    adServerInputStream.flatMap(new
AdServerLogRichFlatMap()).name("WriteServerContext");
```

AdServerLogRichFlatMap

从kafka读取ad_server_log，把数据写入到 redis和hbase

```
public class AdServerLogRichFlatMap extends RichFlatMapFunction<AdServerLog,
String> {
    String tableName = Constants.TABLE_NAME;
    HTable hTable;
    Jedis jedis;

    int expire = 1 * 24 * 60 * 60;

    @Override
    public void open(Configuration parameters) throws Exception {
        hTable = HBaseUtils.initHbaseClient(tableName);
        jedis = RedisUtils.initRedis();
        super.open(parameters);
    }

    @Override
    public void flatMap(AdServerLog adServerLog, Collector<String> collector)
throws Exception {
        byte[] key = ETLUtils.generateBytesKey(adServerLog);
        AdServerLog context = ETLUtils.generateContext(adServerLog);
        ETLUtils.writeRedis(jedis, key, context);
        ETLUtils.writeHbase(hTable, key, context);
    }
}
```

HBaseUtils

```
package com.nx.stream.utils;



import java.io.*;
import java.util.*;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.hbase.*;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.filter.FilterList;
import org.apache.hadoop.hbase.util.*;

public class HBaseUtils {

    private static Connection connection;

    private static Configuration configuration;

    static {
        configuration = HBaseConfiguration.create();
        configuration.set("hbase.zookeeper.property.clientPort", "2181");
        // 如果是集群 则主机名用逗号分隔
        configuration.set("hbase.zookeeper.quorum", "10.0.0.9");
        try {
            connection = ConnectionFactory.createConnection(configuration);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static HTable initHbaseClient(String tableName) {
        try {
            return new HTable(configuration, tableName);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }

    /**
     * 创建 HBase 表
     *
     * @param tableName      表名
     * @param columnFamilies 列族的数组
     */
    public static boolean createTable(String tableName, List<String>
columnFamilies) {
        try {
            HBaseAdmin admin = (HBaseAdmin) connection.getAdmin();
            if (admin.tableExists(tableName)) {
                return false;
            }
            HTableDescriptor tableDescriptor = new
HTableDescriptor(TableName.valueOf(tableName));
            columnFamilies.forEach(columnFamily -> {
                HColumnDescriptor columnDescriptor = new
HColumnDescriptor(columnFamily);
                columnDescriptor.setMaxVersions(1);
                tableDescriptor.addFamily(columnDescriptor);
            });
            admin.createTable(tableDescriptor);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return true;
    }


    /**
     * 删除 hBase 表
     *
     * @param tableName 表名
     */
    public static boolean deleteTable(String tableName) {
        try {
            HBaseAdmin admin = (HBaseAdmin) connection.getAdmin();
            // 删除表前需要先禁用表
            admin.disableTable(tableName);
            admin.deleteTable(tableName);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return true;
    }

    /**
     * 插入数据
     *
     * @param tableName        表名
     * @param rowKey           唯一标识
     * @param columnFamilyName 列族名
     * @param qualifier        列标识
     * @param value            数据
     */
    public static boolean putRow(String tableName, String rowKey, String
columnFamilyName, String qualifier,
                                 String value) {
        try {
            Table table = connection.getTable(TableName.valueOf(tableName));
            Put put = new Put(Bytes.toBytes(rowKey));
            put.addColumn(Bytes.toBytes(columnFamilyName),
Bytes.toBytes(qualifier), Bytes.toBytes(value));
            table.put(put);
            table.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return true;
    }


    /**
     * 根据 rowKey 获取指定行的数据
     *
     * @param tableName 表名
     * @param rowKey    唯一标识
     */
    public static Result getRow(String tableName, String rowKey) {
        try {
            Table table = connection.getTable(TableName.valueOf(tableName));
            Get get = new Get(Bytes.toBytes(rowKey));
            return table.get(get);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }


    /**
     * 获取指定行指定列 (cell) 的最新版本的数据
     *
     * @param tableName    表名
     * @param rowKey       唯一标识
     * @param columnFamily 列族
     * @param qualifier    列标识
     */
    public static String getCell(String tableName, String rowKey, String
columnFamily, String qualifier) {
        try {
            Table table = connection.getTable(TableName.valueOf(tableName));
            Get get = new Get(Bytes.toBytes(rowKey));
            if (!get.isCheckExistenceOnly()) {
                get.addColumn(Bytes.toBytes(columnFamily),
Bytes.toBytes(qualifier));
                Result result = table.get(get);
                byte[] resultValue =
result.getValue(Bytes.toBytes(columnFamily), Bytes.toBytes(qualifier));
                return Bytes.toString(resultValue);
            } else {
                return null;
            }

        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }


    /**
     * 检索全表
     *
     * @param tableName 表名
     */
    public static ResultScanner getScanner(String tableName) {
        try {
            Table table = connection.getTable(TableName.valueOf(tableName));
            Scan scan = new Scan();
            return table.getScanner(scan);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }


    /**
     * 检索表中指定数据
     *
     * @param tableName  表名
     * @param filterList 过滤器
     */

    public static ResultScanner getScanner(String tableName, FilterList
filterList) {
        try {
            Table table = connection.getTable(TableName.valueOf(tableName));
            Scan scan = new Scan();
            scan.setFilter(filterList);
            return table.getScanner(scan);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }

    /**
     * 检索表中指定数据
     *
     * @param tableName   表名
     * @param startRowKey 起始 RowKey
     * @param endRowKey   终止 RowKey
     * @param filterList  过滤器
     */

    public static ResultScanner getScanner(String tableName, String
startRowKey, String endRowKey,
                                           FilterList filterList) {
        try {
            Table table = connection.getTable(TableName.valueOf(tableName));
            Scan scan = new Scan();
            scan.setStartRow(Bytes.toBytes(startRowKey));
            scan.setStopRow(Bytes.toBytes(endRowKey));
            scan.setFilter(filterList);
            return table.getScanner(scan);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }

    /**
     * 删除指定行记录
     *
     * @param tableName 表名
     * @param rowKey    唯一标识
     */
    public static boolean deleteRow(String tableName, String rowKey) {
        try {
            Table table = connection.getTable(TableName.valueOf(tableName));
            Delete delete = new Delete(Bytes.toBytes(rowKey));
            table.delete(delete);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return true;
    }
    /**
     * 删除指定行的指定列
     *
     * @param tableName  表名
     * @param rowKey     唯一标识
     * @param familyName 列族
     * @param qualifier  列标识
     */
    public static boolean deleteColumn(String tableName, String rowKey, String
familyName,
                                       String qualifier) {
        try {
            Table table = connection.getTable(TableName.valueOf(tableName));
            Delete delete = new Delete(Bytes.toBytes(rowKey));
            delete.addColumn(Bytes.toBytes(familyName),
Bytes.toBytes(qualifier));
            table.delete(delete);
            table.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return true;
    }

}
```

```
        <dependency>
            <groupId>org.apache.hbase</groupId>
            <artifactId>hbase-shaded-client</artifactId>
            <version>1.2.0</version>
        </dependency>
```

RedisUtils

```
package com.nx.stream.utils;

import redis.clients.jedis.Jedis;
import redis.clients.jedis.JedisPool;
import redis.clients.jedis.JedisPoolConfig;

import java.time.Duration;

public class RedisUtils {

    static JedisPool jedisPool;

    static {
        final JedisPoolConfig poolConfig = new JedisPoolConfig();
        poolConfig.setMaxTotal(128);
        poolConfig.setMaxIdle(128);
        poolConfig.setMinIdle(16);
        poolConfig.setTestOnBorrow(true);
        poolConfig.setTestOnReturn(true);
        poolConfig.setTestWhileIdle(true);

poolConfig.setMinEvictableIdleTimeMillis(Duration.ofSeconds(60).toMillis());

poolConfig.setTimeBetweenEvictionRunsMillis(Duration.ofSeconds(30).toMillis())
;
        poolConfig.setNumTestsPerEvictionRun(3);
        poolConfig.setBlockWhenExhausted(true);
        jedisPool = new JedisPool(poolConfig, "10.0.0.9", 6379);
    }

    public static Jedis initRedis() {
        return jedisPool.getResource();
    }
}
```

```
        <dependency>
            <groupId>redis.clients</groupId>
            <artifactId>jedis</artifactId>
            <version>2.8.1</version>
        </dependency>
```

生成唯一key

```
vim com.nx.stream.utils.ETLUtils

    public static byte[] generateBytesKey(AdClientLog context) {
        StringBuilder stringBuilder = new StringBuilder();
        stringBuilder.append(context.getRequestId());
        stringBuilder.append(context.getCreativeId());
        stringBuilder.append(context.getUnitId());
        return stringBuilder.toString().getBytes();
    }

    public static byte[] generateBytesKey(AdServerLog context) {
        StringBuilder stringBuilder = new StringBuilder();
        stringBuilder.append(context.getRequestId());
        stringBuilder.append(context.getCreativeId());
        stringBuilder.append(context.getUnitId());
        return stringBuilder.toString().getBytes();
    }
    public static byte[] generateBytesKey(AdLog context) {
        StringBuilder stringBuilder = new StringBuilder();
        stringBuilder.append(context.getRequestId());
        stringBuilder.append(context.getCreativeId());
        stringBuilder.append(context.getUnitId());
        return stringBuilder.toString().getBytes();
    }
```

生成关联内容context

```
vim com.nx.stream.utils.ETLUtils

    public static AdServerLog generateContext(AdServerLog adServerLog) {
        AdServerLog.Builder context = AdServerLog.newBuilder();
        context.setGender(adServerLog.getGender());
        context.setAge(adServerLog.getAge());
        context.setCountry(adServerLog.getCountry());
        context.setSourceType(adServerLog.getSourceType());
        context.setBidType(adServerLog.getBidType());
        return context.build();
    }
```

将context 写入redis

```
    static final int DEFAULT_EXPIRE = 1 * 60 * 60;
    public static void writeRedis(Jedis jedis, byte[] key, AdServerLog
context) {
        jedis.set(key, context.toByteArray());
        jedis.expire(key, DEFAULT_EXPIRE);
    }
```

将context 写入hbase

```
创建hbase表
cd /work/hadoop
hbase shell
list #查看存在哪些表
create 'context_test', 'cf1'
```

```


    static final byte[] DEFAULT_COLUMN_FAMILY = Bytes.toBytes("cf1");
    static final byte[] DEFAULT_COLUMN_KEY = Bytes.toBytes("v");
    static final String RETRY_TOPIC = Constants.CLIENT_LOG_RETRY;
    public static void writeHbase(HTable hTable, byte[] key, AdServerLog
context) {
        try {
            Put put = new Put(key);
            put.add(DEFAULT_COLUMN_FAMILY, DEFAULT_COLUMN_KEY,
context.toByteArray());
            hTable.put(put);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

3.StreamJoinJob - 处理ad_client_log

```
vim com.nx.stream.ad.StreamJoinJob;
    DataStream<AdLog> adLogStream = adClientInputStream.flatMap(new
AdClientLogRichFlatMap());
```

AdClientLogRichFlatMap

```
package com.nx.stream.ad;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.nx.stream.entity.AdClientLog;
import com.nx.stream.entity.AdLog;
import com.nx.stream.entity.AdServerLog;
import com.nx.stream.utils.*;
import org.apache.flink.api.common.functions.RichFlatMapFunction;
import org.apache.flink.configuration.Configuration;
import org.apache.flink.util.Collector;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.kafka.clients.producer.Producer;
import redis.clients.jedis.Jedis;

public class AdClientLogRichFlatMap extends RichFlatMapFunction<AdClientLog,
AdLog> {
    String tableName = Constants.TABLE_NAME;
    HTable hTable;
    Jedis jedis;
    Producer producer;
    ObjectMapper objectMapper;

    @Override
    public void open(Configuration parameters) throws Exception {
        hTable = HBaseUtils.initHbaseClient(tableName);
        jedis = RedisUtils.initRedis();
        producer = KafkaProducerUtils.getProducer();
        objectMapper = new ObjectMapper();
        super.open(parameters);
    }

    @Override
    public void flatMap(AdClientLog adClientLog, Collector<AdLog> collector)
throws Exception {
        byte[] key = ETLUtils.generateBytesKey(adClientLog);
        AdServerLog context = ETLUtils.getContext(jedis, hTable, key);
        AdLog adLog = ETLUtils.buildAdLog(adClientLog, context);
        if (context == null) {
            ETLUtils.sendRetry(producer, adLog.toByteArray());
        } else {
            collector.collect(adLog);
        }
    }
}
```

```
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>2.11.0</version>
        </dependency>
```

读取context

```
    public static AdServerLog getContext(Jedis jedis, HTable hTable, byte[]
key) {
        AdServerLog context = getContext(jedis, key);
        if (context == null)
            context = getContext(hTable, key);
        return context;
    }
    public static AdServerLog getContext(Jedis jedis, byte[] key) {
        byte[] data = jedis.get(key);
        if (data == null)
            return null;
        try {
            return AdServerLog.parseFrom(data);
        } catch (InvalidProtocolBufferException e) {
            e.printStackTrace();
        }
        return null;
    }

    public static AdServerLog getContext(HTable hTable, byte[] key) {
        Get get = new Get(key);
        try {
            Result data = hTable.get(get);
            if (data == null)
                return null;
            return AdServerLog.parseFrom(data.getValue(DEFAULT_COLUMN_FAMILY,
DEFAULT_COLUMN_KEY));
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }
```

构建ad_log

```

    public static AdLog buildAdLog(AdClientLog adClientLog, AdServerLog
context) {
        AdLog.Builder adLogBuilder = AdLog.newBuilder();
        ProcessInfo.Builder processInfoBuilder = ProcessInfo.newBuilder();
        processInfoBuilder.setProcessTimestamp(System.currentTimeMillis());

        adLogBuilder.setRequestId(adClientLog.getRequestId());
        adLogBuilder.setTimestamp(adClientLog.getTimestamp());
        adLogBuilder.setDeviceId(adClientLog.getDeviceId());
        adLogBuilder.setOs(adClientLog.getOs());
        adLogBuilder.setNetwork(adClientLog.getNetwork());
        adLogBuilder.setUserId(adClientLog.getUserId());
        if (context != null) {
            processInfoBuilder.setJoinServerLog(true);
            joinContext(adLogBuilder, context);
        } else {
            processInfoBuilder.setRetryCount(1);
            processInfoBuilder.setJoinServerLog(false);
        }
        adLogBuilder.setSourceType(adClientLog.getSourceType());
        adLogBuilder.setPosId(adClientLog.getPosId());
        adLogBuilder.setAccountId(adClientLog.getAccountId());
        adLogBuilder.setCreativeId(adClientLog.getCreativeId());
        adLogBuilder.setUnitId(adClientLog.getUnitId());
        switch (adClientLog.getEventType()) {
            case "SEND":
                adLogBuilder.setSend(1);
                break;
            case "IMPRESSION":
                adLogBuilder.setImpression(1);
                break;
            case "CLICK":
                adLogBuilder.setClick(1);
                break;
            case "DOWNLOAD":
                adLogBuilder.setDownload(1);
                break;
            case "INSTALLED":
                adLogBuilder.setInstalled(1);
                break;
            case "PAY":
                adLogBuilder.setPay(1);
                break;
            default:
                break;
        }
        return adLogBuilder.build();
    }

    public static void joinContext(AdLog.Builder adLogBuilder, AdServerLog
context) {
        adLogBuilder.setGender(context.getGender());
        adLogBuilder.setAge(context.getAge());
        adLogBuilder.setCountry(context.getCountry());
        adLogBuilder.setProvince(context.getProvince());
        adLogBuilder.setCity(context.getCity());

        adLogBuilder.setBidPrice(context.getBidPrice());
    }
```

重试

```
vim com.nx.stream.utils.ETLUtils
    public static void sendRetry(Producer producer, byte[] value) {
        sendKafka(producer, RETRY_TOPIC, value);
    }
```

4.StreamJoinJob - sink to kafka

```
    // 添加sink
    Properties producerProperties =
FlinkKafkaProducerUtils.getProducerProperties(BROKERS);
    FlinkKafkaProducer010<AdLog> adLogSink = new FlinkKafkaProducer010<AdLog>
(KAFKA_AD_LOG, new AdLogSchema(), producerProperties);

    adLogStream.addSink(adLogSink).name("AdLogProcesser");
```

FlinkKafkaProducerUtils

```
package com.nx.stream.utils;
```

```
import java.util.Properties;

public class FlinkKafkaProducerUtils {

    public static Properties getProducerProperties(String brokers) {
        Properties properties = getCommonProperties();
        properties.setProperty("bootstrap.servers", brokers);
        properties.setProperty("metadata.broker.list", brokers);
        properties.setProperty("zookeeper.connect",
"10.0.0.9:2181,10.0.0.8:2181");
        return properties;
    }

    public static Properties getCommonProperties() {
        Properties properties = new Properties();
        properties.setProperty("linger.ms", "100");
        properties.setProperty("retries", "100");
        properties.setProperty("retry.backoff.ms", "200");
        properties.setProperty("buffer.memory", "524288");
        properties.setProperty("batch.size", "100");
        properties.setProperty("max.request.size", "524288");
        properties.setProperty("compression.type", "snappy");
        properties.setProperty("request.timeout.ms", "180000");
        properties.setProperty("max.block.ms", "180000");
        return properties;
    }
}
```

AdLogSchema

```
package com.nx.stream.entity.schema;

import com.nx.stream.entity.AdLog;
import org.apache.flink.api.common.serialization.DeserializationSchema;
import org.apache.flink.api.common.serialization.SerializationSchema;
import org.apache.flink.api.common.typeinfo.TypeHint;
import org.apache.flink.api.common.typeinfo.TypeInformation;

import java.io.IOException;

public class AdLogSchema implements DeserializationSchema<AdLog>,
SerializationSchema<AdLog> {
    @Override
    public AdLog deserialize(byte[] bytes) throws IOException {
        return AdLog.parseFrom(bytes);
    }

    @Override
    public boolean isEndOfStream(AdLog adClientLog) {
        return false;
    }

    @Override
    public byte[] serialize(AdLog adClientLog) {
        return new byte[0];
    }

    @Override
    public TypeInformation<AdLog> getProducedType() {
        return TypeInformation.of(new TypeHint<AdLog>() {
        });
    }
}
```

5. 上线运行

```
flink run -m yarn-cluster -c com.nx.stream.ad.StreamJoinJob
/work/hadoop/v1/flink-ad-1.0-SNAPSHOT.jar
```

二. adlog etl

```
    FlinkKafkaProducer010<String> adLogReportSink = new
FlinkKafkaProducer010<String>(KAFKA_AD_LOG_REPORT, new SimpleStringSchema(),
producerProperties);
    adLogStream.flatMap(new
AdLogRichFlatMap()).addSink(adLogReportSink).name("AdLogReport");
    // execute program
    env.execute("Stream join job");
```

1. 创建kafka source

这里为了简便直接把这个任务和 stream join合并在一起

2. AdLogRichFlatMap

```
package com.nx.stream.ad;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.nx.stream.entity.AdClientLog;
import com.nx.stream.entity.AdLog;
import com.nx.stream.entity.AdServerLog;
import com.nx.stream.entity.dto.AdLogDTO;
import com.nx.stream.utils.*;
import org.apache.flink.api.common.functions.RichFlatMapFunction;
import org.apache.flink.configuration.Configuration;
import org.apache.flink.util.Collector;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.kafka.clients.producer.Producer;
import redis.clients.jedis.Jedis;

public class AdLogRichFlatMap extends RichFlatMapFunction<AdLog, String> {
    ObjectMapper objectMapper;

    @Override
    public void open(Configuration parameters) throws Exception {
        objectMapper = new ObjectMapper();
        super.open(parameters);
    }

    @Override
    public void flatMap(AdLog adLog, Collector<String> collector) throws
Exception {
        AdLogDTO adLogDTO = ETLUtils.buildAdLogDTO(adLog);
        collector.collect(objectMapper.writeValueAsString(adLogDTO));
    }
}
```

buildAdLogDTO

```
    public static AdLogDTO buildAdLogDTO(AdLog adLog) {
        AdLogDTO adLogDTO = new AdLogDTO();
        adLogDTO.setRequestId(adLog.getRequestId());
        adLogDTO.setTimestamp(adLog.getTimestamp());
        adLogDTO.setDeviceId(adLog.getDeviceId());
        adLogDTO.setOs(adLog.getOs());
        adLogDTO.setNetwork(adLog.getNetwork());
        adLogDTO.setUserId(adLog.getUserId());
        adLogDTO.setGender(adLog.getGender());
        adLogDTO.setAge(adLog.getAge());
        adLogDTO.setCountry(adLog.getCountry());
        adLogDTO.setProvince(adLog.getProvince());
        adLogDTO.setCity(adLog.getCity());
        adLogDTO.setSourceType(adLog.getSourceType());
        adLogDTO.setBidType(adLog.getBidType());
        adLogDTO.setBidPrice(adLog.getBidPrice());
        adLogDTO.setPosId(adLog.getPosId());
        adLogDTO.setAccountId(adLog.getAccountId());
        adLogDTO.setCreativeId(adLog.getCreativeId());
        adLogDTO.setUnitId(adLog.getUnitId());
        adLogDTO.setEventType(adLog.getEventType());
        adLogDTO.setSend(adLog.getSend());
        adLogDTO.setImpression(adLog.getImpression());
        adLogDTO.setClick(adLog.getClick());
        adLogDTO.setDownload(adLog.getDownload());
        adLogDTO.setInstalled(adLog.getInstalled());
        adLogDTO.setPay(adLog.getPay());
        return adLogDTO;
    }
```

AdLogDTO

```
package com.nx.stream.entity.dto;

import com.fasterxml.jackson.annotation.JsonProperty;
import lombok.Data;

@Data
public class AdLogDTO {
    @JsonProperty("request_id")
    private String requestId;
    private long timestamp;
    @JsonProperty("device_id")
    private String deviceId;
    private String os;
    private String network;
    // dmp;
    @JsonProperty("user_id")
    private String userId;
    private String gender;
    private int age;
    private String country;
    private String province;
    private String city;
    // ad;
    @JsonProperty("source_type")
    private String sourceType;
    @JsonProperty("bid_type")
    private String bidType;
    @JsonProperty("bid_price")
    private long bidPrice;
    @JsonProperty("pos_id")
    private long posId;
    @JsonProperty("account_id")
    private long accountId;
    @JsonProperty("creative_id")
    private long creativeId;
    @JsonProperty("unit_id")
    private long unitId;
    // event;
    @JsonProperty("event_type")
    private String eventType;
    private long send;
    private long impression;
    private long click;
    private long download;
    private long installed;
    private long pay;
}
```

```
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>1.18.10</version>
        </dependency>
```

3. 上线运行

```
flink run -m yarn-cluster -c com.nx.stream.ad.StreamJoinJob
/work/hadoop/v1/flink-ad-1.0-SNAPSHOT.jar
```

三. 重试任务

1. StreamJoinRetryJob - 创建kafka source

```
vim com.nx.stream.ad.StreamJoinRetryJob

    private static String KAFKA_CLIENT_LOG_RETRY = Constants.CLIENT_LOG;
    private static String KAFKA_AD_LOG = Constants.AD_LOG;
    private static String KAFKA_AD_LOG_REPORT = Constants.AD_LOG_REPORT;
    private static String BROKERS = Constants.BROKERS;

    public static void main(String[] args) throws Exception {
        String groupId = "flink-join-retry-test";
        // set up the streaming execution environment
        final StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
        env.enableCheckpointing(60000);
        env.getCheckpointConfig().setFailOnCheckpointingErrors(false);
        env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);
        env.getConfig().enableForceAvro();
        env.getConfig().registerTypeWithKryoSerializer(AdLog.class,
ProtobufSerializer.class);
        Properties properties =
FlinkKafkaConsumerUtils.getConsumerProperties(BROKERS, KAFKA_CLIENT_LOG_RETRY,
groupId);

        DataStreamSource<AdLog> adLogInputStream = env.addSource(
                new FlinkKafkaConsumer010<>(KAFKA_CLIENT_LOG_RETRY, new
AdLogSchema(), properties));
```

2. StreamJoinRetryJob - 重试

```
         DataStream<AdLog> adLogStream =
adLogInputStream.keyBy(AdLog::getRequestId)
                .timeWindow(Time.seconds(2))
                .trigger(PurgingTrigger.of(ProcessingTimeTrigger.create()))
                .apply(new AdLogRetryWindowFunction());
```

AdLogRetryWindowFunction

```
package com.nx.stream.ad;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.nx.stream.entity.AdLog;
import com.nx.stream.entity.AdServerLog;
import com.nx.stream.entity.ProcessInfo;
import com.nx.stream.entity.ProcessInfoOrBuilder;
import com.nx.stream.utils.*;
import org.apache.flink.configuration.Configuration;
import org.apache.flink.streaming.api.functions.windowing.RichWindowFunction;
import org.apache.flink.streaming.api.windowing.windows.TimeWindow;
import org.apache.flink.util.Collector;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.kafka.clients.producer.Producer;
import redis.clients.jedis.Jedis;

import java.util.Iterator;

public class AdLogRetryWindowFunction extends RichWindowFunction<AdLog, AdLog,
String, TimeWindow> {
    String tableName = Constants.TABLE_NAME;
    HTable hTable;
    Jedis jedis;
    Producer producer;
    ObjectMapper objectMapper;

    @Override
    public void open(Configuration parameters) throws Exception {
        hTable = HBaseUtils.initHbaseClient(tableName);
        jedis = RedisUtils.initRedis();
        producer = KafkaProducerUtils.getProducer();
        objectMapper = new ObjectMapper();
        super.open(parameters);
    }


    @Override
    public void apply(String requestId, TimeWindow timeWindow, Iterable<AdLog>
iterable, Collector<AdLog> collector) throws Exception {
        Iterator<AdLog> itr = iterable.iterator();
        while (itr.hasNext()) {
            AdLog adLog = itr.next();
            if (System.currentTimeMillis() -
adLog.getProcessInfo().getProcessTimestamp() > 1000) {
                byte[] key = ETLUtils.generateBytesKey(adLog);
                AdServerLog context = ETLUtils.getContext(jedis, key);
                AdLog.Builder adLogBuilder = adLog.toBuilder();
                ProcessInfo.Builder processBuilder =
adLog.getProcessInfo().toBuilder();
                processBuilder.setRetryCount(processBuilder.getRetryCount() +
1);

processBuilder.setProcessTimestamp(System.currentTimeMillis());
                adLogBuilder.setProcessInfo(processBuilder.build());

                if (context == null) {
                    ETLUtils.sendRetry(producer,
adLogBuilder.build().toByteArray());
                } else {
                    ETLUtils.joinContext(adLogBuilder, context);
                    collector.collect(adLog);
                }

            }
        }
    }
}
```

四. adlog etl

同上面【二】

五. data sink

```
vim com.nx.stream.sink.SinkToHiveJob
```

1. 创建kafka source

```
    private static String KAFKA_SERVER_LOG = Constants.SERVER_LOG;
    private static String KAFKA_CLIENT_LOG = Constants.CLIENT_LOG;
    private static String KAFKA_AD_LOG = Constants.AD_LOG;
    private static String BROKERS = Constants.BROKERS;


        String groupId = "flink-sink-task-test2";
        // set up the streaming execution environment
        final StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
//        final StreamExecutionEnvironment env =
StreamExecutionEnvironment.createLocalEnvironment();
        env.enableCheckpointing(60000);
        env.getCheckpointConfig().setFailOnCheckpointingErrors(false);
        env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);
        env.getConfig().enableForceAvro();
        env.getConfig().registerTypeWithKryoSerializer(AdServerLog.class,
ProtobufSerializer.class);
        env.getConfig().registerTypeWithKryoSerializer(AdClientLog.class,
ProtobufSerializer.class);
        env.getConfig().registerTypeWithKryoSerializer(AdLog.class,
ProtobufSerializer.class);
        Properties properties =
FlinkKafkaConsumerUtils.getConsumerProperties(BROKERS, KAFKA_SERVER_LOG,
groupId);
        Properties properties2 =
FlinkKafkaConsumerUtils.getConsumerProperties(BROKERS, KAFKA_CLIENT_LOG,
groupId);
        Properties properties3 =
FlinkKafkaConsumerUtils.getConsumerProperties(BROKERS, KAFKA_AD_LOG, groupId);
        DataStreamSource<AdServerLog> adServerInputStream = env.addSource(
                new FlinkKafkaConsumer010<>(KAFKA_SERVER_LOG, new
AdServerLogSchema(), properties));
        DataStreamSource<AdClientLog> adClientInputStream = env.addSource(
                new FlinkKafkaConsumer010<>(KAFKA_CLIENT_LOG, new
AdClientLogSchema(), properties2));
```

2. 按照日志时间归档

```
    private static String partition = "'dt='yyyyMMdd/'hour'=HH";

        buildSink(adServerInputStream, KAFKA_SERVER_LOG, new
AdServerEventTimeBucketer(partition));
        buildSink(adClientInputStream, KAFKA_CLIENT_LOG, new
AdClientEventTimeBucketer(partition));
```

BucketingSink

```
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-connector-filesystem_2.11</artifactId>
            <version>${flink.version}</version>
        </dependency>
```

AdServerEventTimeBucketer

```
package com.nx.stream.sink;

import com.nx.stream.entity.AdServerLog;
import org.apache.flink.streaming.connectors.fs.Clock;
import org.apache.flink.streaming.connectors.fs.bucketing.Bucketer;
import org.apache.hadoop.fs.Path;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.time.Instant;
import java.time.ZoneId;
import java.time.format.DateTimeFormatter;

public class AdServerEventTimeBucketer implements Bucketer<AdServerLog> {

    private static final String DEFAULT_FORMAT_STRING = "yyyyMMdd";

    private final String formatString;

    private final ZoneId zoneId;
    private transient DateTimeFormatter dateTimeFormatter;

    public AdServerEventTimeBucketer() {
        this(DEFAULT_FORMAT_STRING);
    }

    public AdServerEventTimeBucketer(String formatString) {
        this(formatString, ZoneId.systemDefault());
    }

    public AdServerEventTimeBucketer(ZoneId zoneId) {
        this(DEFAULT_FORMAT_STRING, zoneId);
    }

    public AdServerEventTimeBucketer(String formatString, ZoneId zoneId) {
        this.formatString = formatString;
        this.zoneId = zoneId;
        this.dateTimeFormatter =
DateTimeFormatter.ofPattern(this.formatString).withZone(this.zoneId);
    }
    //记住，这个方法一定要加，否则dateTimeFormatter对象会是空，此方法会在反序列的时候调
用，这样才能正确初始化dateTimeFormatter对象
    //那有的人问了，上面构造函数不是初始化了吗？反序列化的时候是不走构造函数的
    private void readObject(ObjectInputStream in) throws IOException,
ClassNotFoundException {
        in.defaultReadObject();
        this.dateTimeFormatter =
DateTimeFormatter.ofPattern(formatString).withZone(zoneId);
    }

    @Override
    public Path getBucketPath(Clock clock, Path basePath, AdServerLog
adServerLog) {
        String newDateTimeString =
dateTimeFormatter.format(Instant.ofEpochMilli(adServerLog.getTimestamp()));
        return new Path(basePath + "/" + newDateTimeString);
    }
}
```

AdClientEventTimeBucketer

```
package com.nx.stream.sink;

import com.nx.stream.entity.AdClientLog;
import org.apache.flink.streaming.connectors.fs.Clock;
import org.apache.flink.streaming.connectors.fs.bucketing.Bucketer;
import org.apache.hadoop.fs.Path;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.time.Instant;
import java.time.ZoneId;
import java.time.format.DateTimeFormatter;

public class AdClientEventTimeBucketer implements Bucketer<AdClientLog> {

    private static final String DEFAULT_FORMAT_STRING = "yyyyMMdd";

    private final String formatString;

    private final ZoneId zoneId;
    private transient DateTimeFormatter dateTimeFormatter;

    public AdClientEventTimeBucketer() {
        this(DEFAULT_FORMAT_STRING);
    }
    public AdClientEventTimeBucketer(String formatString) {
        this(formatString, ZoneId.systemDefault());
    }

    public AdClientEventTimeBucketer(ZoneId zoneId) {
        this(DEFAULT_FORMAT_STRING, zoneId);
    }

    public AdClientEventTimeBucketer(String formatString, ZoneId zoneId) {
        this.formatString = formatString;
        this.zoneId = zoneId;
        this.dateTimeFormatter =
DateTimeFormatter.ofPattern(this.formatString).withZone(this.zoneId);
    }

    //记住，这个方法一定要加，否则dateTimeFormatter对象会是空，此方法会在反序列的时候调
用，这样才能正确初始化dateTimeFormatter对象
    //那有的人问了，上面构造函数不是初始化了吗？反序列化的时候是不走构造函数的
    private void readObject(ObjectInputStream in) throws IOException,
ClassNotFoundException {
        in.defaultReadObject();
        this.dateTimeFormatter =
DateTimeFormatter.ofPattern(formatString).withZone(zoneId);
    }

    @Override
    public Path getBucketPath(Clock clock, Path basePath, AdClientLog
adClientLog) {
        String newDateTimeString =
dateTimeFormatter.format(Instant.ofEpochMilli(adClientLog.getTimestamp()));
        return new Path(basePath + "/" + newDateTimeString);
    }
}
```

AdLogEventTimeBucketer

```
package com.nx.stream.sink;

import com.nx.stream.entity.AdLog;
import org.apache.flink.streaming.connectors.fs.Clock;
import org.apache.flink.streaming.connectors.fs.bucketing.Bucketer;
import org.apache.hadoop.fs.Path;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.time.Instant;
import java.time.ZoneId;
import java.time.format.DateTimeFormatter;
```

```
public class AdLogEventTimeBucketer implements Bucketer<AdLog> {

    private static final String DEFAULT_FORMAT_STRING = "yyyyMMdd";

    private final String formatString;

    private final ZoneId zoneId;
    private transient DateTimeFormatter dateTimeFormatter;

    public AdLogEventTimeBucketer() {
        this(DEFAULT_FORMAT_STRING);
    }

    public AdLogEventTimeBucketer(String formatString) {
        this(formatString, ZoneId.systemDefault());
    }

    public AdLogEventTimeBucketer(ZoneId zoneId) {
        this(DEFAULT_FORMAT_STRING, zoneId);
    }

    public AdLogEventTimeBucketer(String formatString, ZoneId zoneId) {
        this.formatString = formatString;
        this.zoneId = zoneId;
        this.dateTimeFormatter =
DateTimeFormatter.ofPattern(this.formatString).withZone(this.zoneId);
    }

    //记住，这个方法一定要加，否则dateTimeFormatter对象会是空，此方法会在反序列的时候调
用，这样才能正确初始化dateTimeFormatter对象
    //那有的人问了，上面构造函数不是初始化了吗？反序列化的时候是不走构造函数的
    private void readObject(ObjectInputStream in) throws IOException,
ClassNotFoundException {
        in.defaultReadObject();
        this.dateTimeFormatter =
DateTimeFormatter.ofPattern(formatString).withZone(zoneId);
    }

    @Override
    public Path getBucketPath(Clock clock, Path basePath, AdLog adlog) {
        String newDateTimeString =
dateTimeFormatter.format(Instant.ofEpochMilli(adlog.getTimestamp()));
        return new Path(basePath + "/" + newDateTimeString);
    }
}
```

buildSink

```
    private static String HDFS_LOG_HOME = Constants.HDFS_LOG_HOME;
    private static String partition = "'dt='yyyyMMdd/'hour'=HH";

public static void config(BucketingSink sink) {
    sink.setUseTruncate(false);
    sink.setBatchSize(1024 * 1024 * 256L); // 256M 一个文件
    sink.setBatchRolloverInterval(30 * 60 * 1000L);
    sink.setInactiveBucketThreshold(3 * 60 * 1000L);// 3分钟不写入就从 in-
progress 转变为 pending
    sink.setInactiveBucketCheckInterval(30 * 1000L);// 30秒钟检查一次多久没有写入
了，用于判断是否从 in-progress 转变为 pending
    sink.setInProgressSuffix(".in-progress");
    sink.setPendingSuffix(".pending");

}

public static void buildSink(DataStreamSource streamSource, String topic,
Bucketer bucketer) {
    BucketingSink stringSink = new BucketingSink<>(HDFS_LOG_HOME + "json/" +
topic);
    //通过这样的方式来实现数据按事件时间分区
    stringSink.setBucketer(bucketer);
    stringSink.setWriter(new ProtobufStringWriter<>());
    config(stringSink);
    streamSource.addSink(stringSink).name(topic + "-JsonSink");

}
```

ProtobufStringWriter

```
package com.nx.stream.sink;

import com.google.protobuf.Message;
import com.googlecode.protobuf.format.JsonFormat;
import org.apache.flink.streaming.connectors.fs.StreamWriterBase;
import org.apache.flink.streaming.connectors.fs.StringWriter;
import org.apache.flink.streaming.connectors.fs.Writer;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;

import java.io.IOException;
import java.nio.charset.Charset;
import java.nio.charset.IllegalCharsetNameException;
import java.nio.charset.UnsupportedCharsetException;
```

```
public class ProtobufStringWriter<T extends Message> extends
StreamWriterBase<T> {
    private static final long serialVersionUID = 1L;

    private String charsetName;

    private transient Charset charset;

    public ProtobufStringWriter() {
        this("UTF-8");
    }

    public ProtobufStringWriter(String charsetName) {
        this.charsetName = charsetName;
    }

    protected ProtobufStringWriter(ProtobufStringWriter<T> other) {
        super(other);
        this.charsetName = other.charsetName;
    }


    @Override
    public void open(FileSystem fs, Path path) throws IOException {
        super.open(fs, path);

        try {
            this.charset = Charset.forName(charsetName);
        }
        catch (IllegalCharsetNameException e) {
            throw new IOException("The charset " + charsetName + " is not
valid.", e);
        }
        catch (UnsupportedCharsetException e) {
            throw new IOException("The charset " + charsetName + " is not
supported.", e);
        }
    }

    @Override
    public void write(T element) throws IOException {
        FSDataOutputStream outputStream = getStream();

outputStream.write(JsonFormat.printToString(element).getBytes(charset));
        outputStream.write('\n');
    }

    @Override
    public Writer<T> duplicate() {
        return new ProtobufStringWriter<>(this);
    }
}
```

```
        <dependency>
            <groupId>com.googlecode.protobuf-java-format</groupId>
            <artifactId>protobuf-java-format</artifactId>
            <version>1.2</version>
        </dependency>
```

```
        env.execute("Stream sink job");
```

3. 上线运行

```
flink run -m yarn-cluster -c com.nx.stream.sink.SinkToHiveJob
/work/hadoop/v1/flink-ad-1.0-SNAPSHOT.jar
```

Hive 任务开发

建表

server_log_v3

```

create external table if not exists server_log_v3(
    `request_id` string,
    `timestamp` bigint,
    `device_id` string,
    `os` string,
    `network` string,
    `user_id` string,
    `gender` string,
    `age` bigint,
    `country` string,
    `province` string,
    `source_type` string,
    `bid_type` string,
    `bid_price` bigint,
    `pos_id` bigint,
    `account_id` bigint,
    `unit_id` bigint,
    `creative_id` bigint,
    `event_type` string
)comment ""
partitioned by (dt string, hour string)
```

```
row format serde 'org.apache.hive.hcatalog.data.JsonSerDe'
LOCATION
  'hdfs:///home/json/server_log_v3';
msck repair table server_log_v3;
```

client_log_v3

```
create external table if not exists client_log_v3(
    `request_id` string,
    `timestamp` bigint,
    `device_id` string,
    `os` string,
    `network` string,
    `user_id` string,
    `source_type` string,
    `bid_type` string,
    `pos_id` bigint,
    `account_id` bigint,
    `unit_id` bigint,
    `creative_id` bigint,
    `event_type` string
)comment ""
partitioned by (dt string, hour string)
row format serde 'org.apache.hive.hcatalog.data.JsonSerDe'
LOCATION
  'hdfs:///home/json/client_log_v3';

msck repair table client_log_v3;
```

ad_log_v3

```
create external table if not exists ad_log_v3(
    `request_id` string,
    `timestamp` bigint,
    `device_id` string,
    `os` string,
    `network` string,
    `user_id` string,
    `source_type` string,
    `bid_type` string,
    `pos_id` bigint,
    `account_id` bigint,
    `unit_id` bigint,
    `creative_id` bigint,
    `event_type` string
)comment ""
partitioned by (dt string, hour string)
row format serde 'org.apache.hive.hcatalog.data.JsonSerDe'
LOCATION
  'hdfs:///home/json/client_log_v3';

msck repair table client_log_v3;
```

Server_log_context

```
CREATE TABLE IF NOT EXISTS `server_log_context_hi` (
    request_id string COMMENT '请求id',
    account_id bigint comment '账户id',
    unit_id bigint COMMENT '单元id',
    creative_id bigint COMMENT '创意id',
    `timestamp` bigint COMMENT '时间戳',
    gender string COMMENT '',
    age string COMMENT '',
    country string COMMENT '',
    source_type string COMMENT '',
    bid_type string COMMENT ''
)
COMMENT "server log 广告下发信息的日志"
PARTITIONED BY (dt string, hour string)
STORED AS PARQUET;

INSERT OVERWRITE TABLE server_log_context_hi PARTITION (dt='20200923', hour =
'20')
SELECT
  request_id,
  account_id,
  unit_id,
  creative_id,
  `timestamp`,
  gender,
  age,
  country,
  source_type,
  bid_type
FROM server_log_v1 WHERE dt='20200923' AND hour='20'
```

ad_log_joined_hi

```
CREATE TABLE IF NOT EXISTS `server_log_joined_hi` (
    `request_id` string,
    `timestamp` bigint,
    `device_id` string,
    `os` string,
    `network` string,
    `user_id` string,
    `source_type` string,
    `bid_type` string,
    `pos_id` bigint,
    `account_id` bigint,
    `unit_id` bigint,
    `creative_id` bigint,
    `event_type` string
)
COMMENT "拼接后的日志"
PARTITIONED BY (dt string, hour string)
```

测试

```
add jar hdfs:///home/udf/hive-hcatalog-core-3.1.0.jar;
select count(*) from server_log_test where dt='20200728'
```